University of British Columbia
CPSC 111, Intro to Computation
Jan-Apr 2006

Tamara Munzner

Arrays and Class Design

Lecture 18, Tue Mar 14 2006

based on slides by Kurt Eiselt

http://www.cs.ubc.ca/~tmm/courses/cpsc111-06-spr

News

- Midterm 2: Thu Mar 16, 6:30pm
  - Woodward 1&2
  - hour-long exam, reserve 6:30-8 time slot
    - for buffer in case of fire alarms etc
- Assignment 2 was due Friday 5pm
  - Internet crash at 4:45pm led to 24-hr extension to 5pm Sat
  - hardcopy was due Mon noon
- no labs/tutorials this week
  - but one TA will be in lab during normal lab hours to answer questions

Midterm 2

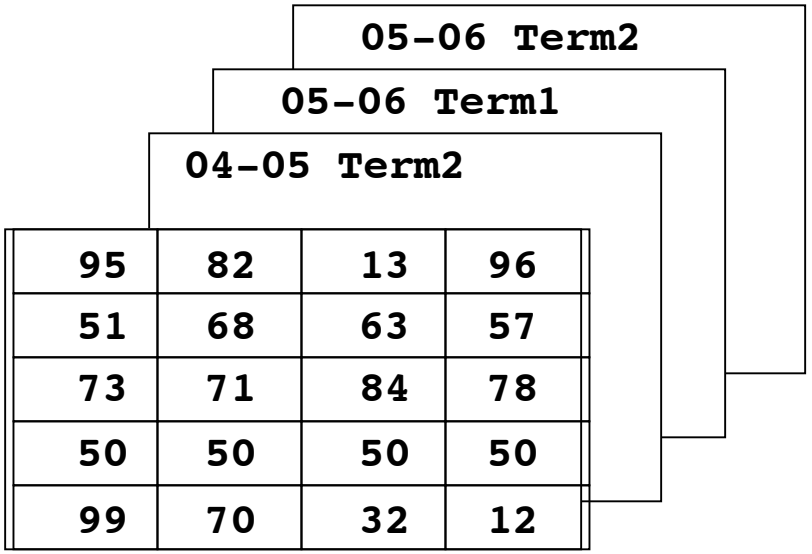
- coverage: through arrays (Chap 8)
  - includes/builds on material covered in previous midterm
- reading
  - Chap 1, 2, 3, 4, 6, 7 (not 5)
  - Chap 8.1, 8.5-8.7
- study tips
  - write and test programs, not just read book
    - try programming exercises from book!

Reading

- This week: 9.3-9.4, 9.6-9.8

Recap/Correct: Multidimensional Arrays

- now that we know 2D, we can do nD!
  - any number of dimensions: 3D, 4D…
    - up to 255D, actually
- example: student quiz scores over multiple terms
  - row: students
  - col: quiz scores
  - stack: term



Recap: Sorting in N^2 Time

Let's assume that your computer could make 1 billion (1,000,000,000) comparisons per second. That's a lot of comparisons in a second. And let's say your computer was using selection sort to sort the names of the people in the following hypothetical telephone books. Here's some mathematical food for thought.

phone book	number of people (N)	N^2	number of seconds needed to sort	
Vancouver	544,320	296,284,262,400	296	or 5 minutes
Canada	30,000,000	900,000,000,000,000	900,000	or 10.4 days
People's Republic of China	1,000,000,000	1,000,000,000,000,000,000	1,000,000,000	or 31.7 years
World	6,000,000,000	36,000,000,000,000,000,000	36,000,000,000	or 1142 years

Bunny Class Warmup

Question 4: **[15 marks]**

Now let's use Java to simulate bunnies! (Why? Because everybody likes bunnies!) In our simulation, each bunny is on a grid at some location defined by an X-coordinate and a Y-coordinate. Also, each bunny has some number of energy units measured in carrot sticks. (X-coordinates, Y-coordinates, and the number of carrot sticks are integer values.) Bunnies can hop north, south, east, or west. When a bunny hops to the north, the bunny's Y-coordinate is increased by 1, and the X-coordinate remains unchanged. When a bunny hops to the west, the bunny's X-coordinate is decreased by 1, and the Y-coordinate remains unchanged. Same idea for hops east (X-coordinate increased by 1, Y-coordinate unchnged) and south (Y-coordinate decreased by 1, X-coordinate unchanged. Note that making one hop requires a bunny to eat one carrot stick, and when a bunny has eaten all of his or her carrot sticks, that bunny can 't hop.

Use Java to create a Bunny class which can be used to generate Bunny objects that behave as described above. When a new Bunny object is created, the Bunny always starts at coordinates X = 10, Y = 10, and the Bunny has 5 carrot sticks. Your Bunny class definition must include a hop(int direction) method, and a displayInfo() method. The direction parameter is 12 for north, 3 for east, 6 for south, and 9 for west (like a clock face). The hop() method should test to make sure that the Bunny has not eaten all the carrot sticks – if the Bunny still has carrot sticks, the hop() method should update coordinates as explained above and print the message "hop". If no carrot sticks remain, it should just print the message "This bunny can't hop".

The displayInfo() method should print the Bunny's location and number of remaining carrot sticks. Below is a simple test program that could be used to test your Bunny class definition, followed by the output we'd expect to see when using this test program with your Bunny class definition.

```
public class BunnyTest
{
  public static void main(String[] args)
  {
    System.out.println("Testing Peter");
    Bunny peter = new Bunny();
    peter.displayInfo();
    peter.hop(12);
    peter.hop(12);
    peter.hop(9);
    peter.displayInfo();
    System.out.println("Testing Emily");
    Bunny emily = new Bunny();
    emily.displayInfo();
    emily.hop(9);
    emily.hop(9);
    emily.hop(9);
    emily.hop(12);
    emily.hop(9);
    emily.hop12();
    emily.displayInfo();
  }
}
```

```
> java BunnyTest
Testing Peter
This bunny is at position 10,10
This bunny has 5 carrot sticks remaining
hop
hop
hop
This bunny is at position 9,12
This bunny has 2 carrot sticks remaining
Testing Emily
This bunny is at position 10,10
This bunny has 5 carrot sticks remaining
hop
hop
hop
hop
hop
This bunny can't hop
This bunny is at position 6,11
This bunny has 0 carrot sticks remaining
>
```

More Bunnies

How could we keep track of a herd of bunnies?

We could make an array of bunnies.

More Bunnies

```
public class BunnyTest1
{
  public static void main (String[] args)
  {
    Bunny[] myBunnyHerd = new Bunny[10];

    myBunnyHerd[0] = new Bunny(3,6,4,"Foofoo");
    myBunnyHerd[1] = new Bunny(7,4,2,"Peter");
    myBunnyHerd[3] = new Bunny(9,2,3,"Ed");

    for(int i = 0; i < myBunnyHerd.length; i++)
    {
      if (myBunnyHerd[i] != null)
      {
        myBunnyHerd[i].hop(3);
        System.out.println(myBunnyHerd[i]);
      }
    }
  }
}
```

Even More Bunnies

Question 5: **[16 marks]**
The world desperately needs better bunny management software, so please help by writing a BunnyHerd class. A BunnyHerd object holds an array of Bunny objects. Your BunnyHerd class definition should include the following four methods:

constructor Expects two parameters, an integer representing the maximum number of bunnies in the herd, and a String for the name of the herd.

`addBunny(int xPos, int yPos, int carrots,String name)` Expects four parameters, the X- and Y-coordinates of the bunny, the number of carrots, and the name. This method creates a new Bunny object and stores the reference to the object in the next available location in the BunnyHerd object.

`deleteBunny(String name)` Expects one parameter, the name of the bunny. This method removes from the BunnyHerd object all references to bunnies with the given `name` by overwriting those references with the `null` pointer. This method does not change the pointer to the next available location in the BunnyHerd object.

`printHerd()` This method uses the `toString()` method of the Bunny object to print information about every Bunny in the herd.

Favorite Colors (If Time…)

- record everybody's favorite color
- how can we do "averages" per row?